ARM Cortex™-A15 MPCore - NEON

Product Revision r2

Software Developers Errata Notice

Non-Confidential - Released

This document contains all software developer errata known at the date of issue in releases r2p0 up to and including revision r2p4

ARM®

Software Developers Errata Notice

Non-Confidential Proprietary Notice

Web Address

http://www.arm.com

Feedback on content

If you have any comments on content, then send an e-mail to errata@arm.com . Give:

- the document title
- the document number, ARM-EPM-028090
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Release Information

Errata are listed in this section if they are new to the document, or marked as "updated" if there has been any change to the erratum text in Chapter 2. Fixed errata are not shown as updated unless the erratum text has changed. The summary table in section 2.2 identifies errata that have been fixed in each product revision.

21 Mar 2013: Changes in Document v4

Page	Status	ID	Cat	Rare	Summary of Erratum
16	New	801819	CatA	Rare	An eviction from L1 data cache might stall indefinitely in the L2 write buffer preventing a snoop from completing
24	Updated	798181	CatB		Moving a virtual page that is being accessed by an active process can lead to unexpected behavior

31 Jan 2013: Changes in Document v3

There are no New or Updated errata in this version.

31 Jan 2013: Changes in Document v2

Page	Status	ID	Cat	Rare	Summary of Erratum
13	New	799271	CatA		A combination of Non-cacheable or streaming writes, a DSB instruction, and a snoop can cause a core to deadlock
26	New	798870	CatB		A memory read can stall indefinitely in the L2 cache
27	New	799270	CatB		Writing ACTLR.SMP when the L2 cache has been idle for an extended period may not work correctly
24	New	798181	CatB	Rare	Moving a virtual page that is being accessed by an active process can lead to unexpected behavior
29	Updated	763126	CatB	Rare	Three processor exclusive access livelock

23 Oct 2012: Changes in Document v1

Page	Status	ID	Cat	Rare	Summary of Erratum
12	New	794724	CatA		L2 Cache initialization sequence may not function correctly if the L2 tag RAM requires two-cycle setup time
29	Updated	763126	CatB	Rare	Three processor exclusive access livelock

Contents

Chapter 1.
Introduction

This chapter introduces the errata notice for the ARM Cortex-A15 processor.

1.1. Scope of this document

This document describes errata categorized by level of severity. Each description includes:

- the current status of the defect
- where the implementation deviates from the specification and the conditions under which erroneous behavior occurs
- the implications of the erratum with respect to typical applications
- the application and limitations of a 'work-around' where possible

This document describes errata that may impact anyone who is developing software that will run on implementations of this ARM product.

1.2. Categorization of errata

Errata recorded in this document are split into the following levels of severity:

Table 1 Categorization of errata

Errata Type	Definition
Category A	A critical error. No workaround is available or workarounds are impactful. The error is likely to be common for many systems and applications.
Category A(rare)	A critical error. No workaround is available or workarounds are impactful. The error is likely to be rare for most systems and applications. Rare is determined by analysis, verification and usage.
Category B	A significant error or a critical error with an acceptable workaround. The error is likely to be common for many systems and applications.
Category B(rare)	A significant error or a critical error with an acceptable workaround. The error is likely to be rare for most systems and applications. Rare is determined by analysis, verification and usage.
Category C	A minor error.

Chapter 2.
Errata Descriptions

2.1. Product Revision Status

The r*n*p*n* identifier indicates the revision status of the product described in this book, where:

r*n* Identifies the major revision of the product.

p*n* Identifies the minor revision or modification status of the product.

2.2. Revisions Affected

Table 2 below lists the product revisions affected by each erratum. A cell marked with **X** indicates that the erratum affects the revision shown at the top of that column.

This document includes errata that affect revision r2 only.

Refer to the reference material supplied with your product to identify the revision of the IP.

Table 2 Revisions Affected

ID	Cat	Rare	Summary of Erratum	r2p0	r2p1	r2p2	r2p3	r2p4
770821	CatA		A15 may block snoops if multi-part DVM message on AR channel has received one but not both R channel responses	X				
781819	CatA		ACE snoop collision with internal A15 snoop may deadlock core	X	X	X		
794724	CatA		L2 Cache initialization sequence may not function correctly if the L2 tag RAM requires two-cycle setup time	X	X	X	X	X
799271	CatA		A combination of Non-cacheable or streaming writes, a DSB instruction, and a snoop can cause a core to deadlock	X	X	X	X	X
773769	CatA	Rare	Large data RAM latencies can lead to rare data corruption	X				
775619	CatA	Rare	L2 may deadlock or corrupt data if all ways in a single set become blocked from replacement	X	X			
801819	CatA	Rare	An eviction from L1 data cache might stall indefinitely in the L2 write buffer preventing a snoop from completing	X	X	X	X	X
774569	CatB		Watchpoint may not be taken on second half of unaligned ld/st crossing a 64-byte aligned boundary	X	X	X	X	X
774769	CatB		Data corruption may occur with store streaming in a system	X	X			
775621	CatB		An eviction behind a store to the on-chip GIC may cause a deadlock in a coherent ACE system	X	X			
777770	CatB		ESR incorrect for AdvSID HCPTR trapped inst in Hyp mode	X	X	X	X	X
784420	CatB		Speculative instruction fetches with MMU disabled might not comply with architectural requirements	X	X	X	X	X

ID	Cat	Rare	Summary of Erratum	r2p0	r2p1	r2p2	r2p3	r2p4
784477	CatB		CTIINTACK register needs clearing each time it is set	X	X	X	X	X
785769	CatB		Undefined exception is not generated for LDC/STC instructions which access DCC registers in User mode	X	X	X	X	X
798181	CatB		Moving a virtual page that is being accessed by an active process can lead to unexpected behavior	X	X	X	X	X
798870	CatB		A memory read can stall indefinitely in the L2 cache	X	X	X	X	X
799270	CatB		Writing ACTLR.SMP when the L2 cache has been idle for an extended period may not work correctly	X	X	X	X	X
763126	CatB	Rare	Three processor exclusive access livelock	X	X	X	X	X
771173	CatB	Rare	Unaligned VLDM larger than 68 bytes to shared cacheable memory hit by snoop can stall core until next interrupt	X				
773319	CatB	Rare	Unaligned page boundary crossing load hit by TLB invalidate can stall until next interrupt	X				
770320	CatC		Single bit ECC error can cause cache maintenance operation to violate memory ordering	X	X	X	X	X
773023	CatC		Order may not be maintained between Strongly Ordered memory requests and Device memory requests on ACE/AXI	X	X	X	X	X
774570	CatC		Fault Status bit in register DBGDSCR is implemented as sticky	X	X	X	X	X
774571	CatC		Sampling registers DBGCIDSR and DBGVIDSR can return incorrect value	X	X	X	X	X
775620	CatC		Unaligned load crossing page boundary between different memory types may stall until next interrupt	X	X	X	X	X
775622	CatC		Debug version of Cache Type Register (CTR) IminLine field are incorrectly encoded	X	X	X	X	X
777769	CatC		ICache parity error may not be corrected for NC code	X	X	X	X	X
777771	CatC		Ordering of read accesses to the same memory location may not be ensured in the case of an unaligned load	X	X			
777772	CatC		Device LDM may stall if memory type changed asynchronously from Device to Normal	X	X	X	X	X
777774	CatC		DCCMVAC/DCCSW colliding with ReadOnce from ACE could cause data corruption	X	X	X	X	X
780121	CatC		PTM might not acknowledge a trace flush request when cpu is in WFI.	X	X	X	X	X
784419	CatC		An unaligned load crossing a 4K boundary between SO/Dev memory and WB memory can stall the core until the next interrupt	X	X	X	X	X
784469	CatC		CTI Authentication Status register is incorrect	X	X	X	X	X
788419	CatC		If a single memory address is aliased as both shareable and non-shareable, a CMO to this same line might deadlock	X	X	X	X	X

2.3. r2p2, r2p3, r2p4 implementation fix

Note the following erratum may be fixed in some implementations of r2p2, r2p3 and r2p4. This can be determined by reading the REVIDR register where a set bit indicates that the erratum is fixed in this part.

REVIDR[0]	794724 L2 Cache initialization sequence may not function correctly if the L2 tag RAM requires two-cycle setup time
REVIDR[1]	799271 A combination of Non-cacheable or streaming writes, a DSB instruction, and a snoop can cause a core to deadlock
REVIDR[2]	798870 A memory read can stall indefinitely in the L2 cache
REVIDR[3]	801819 An eviction from L1 data cache might stall indefinitely in the L2 write buffer preventing a snoop from completing

REVIDR[4]	798181 Moving a virtual page that is being accessed by an active process can lead to unexpected behavior

Note that there is no change to the MIDR which remains at the primary revision r2p2, r2p3 or r2p4, but the REVIDR might be updated from 0x00 to indicate that one or more errata are corrected as shown above. Software will identify this release through the combination of MIDR and REVIDR..

2.4. Category A

770821: A15 may block snoops if multi-part DVM message on AR channel has received one but not both R channel responses

Category A

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0

Description

If the A15 issues a multi-part DVM message on ACE (for example, a TLB invalidate operation by MVA), two separate AR channel requests with the same ARID will be issued. If the interconnect responds to one of the two packets, the L2 buffer will start processing the completion of that DVM command.

If at that point a snoop or DVM message comes in on the AC channel which has an address that matches the address operand of the original multi-part DVM message, that AC channel request will be stalled until the multi-part DVM message completes.

If the response for the second half of the multi-part DVM message is blocked in the interconnect with a dependency on the completion of the AC channel request, the system will deadlock. A15 should not be stalling the AC channel request in this case.

CCI-400 can trigger this erratum on A15, as it can respond to one half of a multi-part DVM and hold off the second half pending an AC channel completion.

Note: This erratum matches bug #5232 in the ARM internal Jira database.

Conditions

1) ACE System
2) DVM operations being issued by A15 to another A15 or A7 cluster, or to a System MMU
3) Multi-part DVM issued (TLB, instruction cache, or BTB invalidate with address)
4) The ACE interconnect responds to only the first half of the multi-part DVM request
5) An ACE channel request is issued to A15 for which the ACADDR field matches the ARADDR

Implications

For multi-cluster ACE systems, or systems with ACE, an A15 cluster, and a peripheral with a System MMU block, DVM operations are required for TLB maintenance and instruction cache maintenance. Multi-part DVM operations are common. This erratum is expected to be rare, but can cause those systems to hang if the right address hazard and system congestion conditions occur.

Workaround

There is no known workaround.

781819: ACE snoop collision with internal A15 snoop may deadlock core

Category A

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2

Description

If three transactions occur to the same L1 data cache set (physical address bits [13:6] are identical) with correct timing, it is possible that A15 will deadlock. An A15 core (coreX) receives an external ACE snoop for a line in its L1 data. That line is targeted for replacement by an outstanding read from coreX. Another read from coreX is transferring data from a second A15 core (coreY). The snoop logic will block further data transfer to the coreX to prevent the first outstanding read from replacing the snoop target while the snoop is being processed by coreX. This prevents the read from coreY from completing. A hazard in the L1 data cache of coreX prevents the external ACE snoop from completing until the read from coreY completes. The snoop is therefore never completed and the A15 deadlocks.

Note: This erratum matches bug #5351 in the ARM internal Jira database.

Configurations Affected:

A15 with 2 or more cores which is receiving ACE snoops

Conditions

1) An external ACE snoop is received for cache line A that is in the L1 data cache of coreX in A15
2) A read from coreX is outstanding to cache line B that will replace cache line A in the L1
3) A read from coreX is outstanding to cache line C that hits unique in another core (coreY) in A15
4) Lines A/B/C are all to the same L1 set (physical_address[13:6] are the same)
5) The external ACE snoop is sent to coreX when:
    - The read for line B has not yet returned any data
    - The read for line C has transferred some but not all of the data from the other A15 core

Implications

If the above conditions occur, the A15 will deadlock.

Workaround

There is no workaround in a coherent ACE system.

794724: L2 Cache initialization sequence may not function correctly if the L2 tag RAM requires two-cycle setup time

Category A

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

If a Cortex-A15 MPCore implementation uses an L2 tag RAM that requires a two cycle setup time for address and data, the L2 cache might not be correctly invalidated by the hardware initialization sequence that occurs after the deassertion of nL2RESET.

By default, the Cortex-A15 MPCore L2 tag RAM input paths are single cycle paths for both setup and hold. However, if the L2 tag RAM used in an implementation requires it, software can program L2CTLR[9] to 1 to configure the setup paths to be two cycle multicycle paths. These must be set correctly before the data cache is enabled.

Cortex-A15 MPCore initializes the L2 cache in hardware when the L2 is reset. Because this hardware initialization occurs before the L2CTLR register can be programmed by software, the hardware sequence must assume tag RAM array timings that will work with all legal RAM instances.

During hardware RAM initialization of the L2 on affected versions of Cortex-A15 MPCore the setup used for the L2 tag RAM is a single cycle. It should be two cycles to support RAMs that require more setup.

Note: This erratum matches bug #5403 in the ARM internal Jira database.

Configurations Affected

This erratum affects implementations that require L2CTLR[9] to be set to 1 because the L2 tag RAM requires two cycle setup.

This erratum does not affect the processor if the revision and variant reported by the MIDR is r2p0, r2p1, r2p2, r2p3 or r2p4 and the REVIDR[0] is set to 1.

Conditions

This erratum requires the following condition:

- The implementation uses an L2 tag RAM that requires two cycle setup on address and data. Affected implementations will program L2CTLR[9]=1 before enabling the data cache.

Implications

The L2 Cache may not be correctly initialized after deassertion of nL2RESET.

1) Valid lines with unknown addresses and data might be present in the cache after reset. If the valid lines match any address in physical memory that is accessed before the lines are evicted, they could deliver incorrect data on reads.

2) If the lines appear valid and dirty, they could generate spurious memory transactions that would corrupt memory or generate errors in the system.

Workaround

Using a lower frequency such that the tag RAM instance can meet timing without the two cycle setup is a valid system workaround. The lower frequency need only be used between deassertion of nL2RESET and the completion of the cache initialization sequence. Software can ensure that the hardware initialization sequence has completed by executing any data cache maintenance operation (e.g. DCCISW) followed by a DSB. The maintenance operation will not complete until the L2 initialization is complete.

Software initialization of the L2 cache is not a valid workaround. The DCISW invalidate by set/way instruction is treated by A15 as a DCCISW clean/invalidate by set/way instruction. At the end of a software initialization of the L2 cache, the cache would be correctly invalid, but the software initialization could cause spurious evictions of lines incorrectly marked as valid and dirty.

799271: A combination of Non-cacheable or streaming writes, a DSB instruction, and a snoop can cause a core to deadlock

Category A

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

If a certain combination of snoops, DSB instruction execution, and stores occurs with a particular timing, a Cortex-A15 MPCore CPU can deadlock. The deadlocked CPU will be unable to complete an eviction until earlier writes complete, and those writes will not be able to complete until that eviction completes.

Note: This erratum matches bug #5416 in the ARM internal Jira database.

Configurations Affected

This erratum does not affect the processor if the revision and variant reported by the MIDR is r2p0, r2p1, r2p2, r2p3 or r2p4 and REVIDR[1] is set to 1.

Conditions

1) One core (coreX) is executing store instructions that are not allocating to the L1 cache. These will be a combination of:
   - Writes to Non-cacheable, Strongly-ordered, or Device memory locations.
   - Write-streaming full line cacheable writes.
2) These stores back up in the memory system and allocate 12 entries of the L2 Write Request Queue
3) CoreX begins the eviction of a cache line (A) that is being replaced by a returning cache line fill
4) Another core (coreY) executes a DSB instruction that generates a TLB synchronization request to coreX
5) A memory request occurs (from coreZ, an ACP request, an external snoop, or a translation table walk request) that
   - hits the cache line (A) being evicted from coreX, OR
   - hits another line in the L1 data cache of coreX that is in uniqueClean or uniqueDirty state.

Implications

If the above scenario occurs with a particular timing, it is possible that the eviction from coreX cannot complete until another L2 write buffer credit is returned, and no such credit can be returned until the eviction completes. In this situation, the processor deadlocks.

In a single core configuration without ACE, the erratum cannot not occur.

In a single core configuration with ACE, but with no other ARM cores capable of issuing a “DVM Sync” request, the erratum cannot occur.

In a two core configuration without ACE, the erratum is believed to be very rare.

In a two core configuration with ACE, or a three or more core configuration, the erratum can occur.

Workaround

There is no known workaround.

2.5. Category A (Rare)

773769: Large data RAM latencies can lead to rare data corruption

Category A Rare
Products Affected: Cortex-A15 MP Core -NEON.
Present in: r2p0

Description

In a system with ACP in use, with long L2 data RAM latencies (4-8 cycles), or with an L2 data RAM slice configured, it is possible that a rare collision between non-cacheable stores and L1 data cache evictions can lead to data corruption.

Note: This erratum matches bug #5269 in the ARM internal Jira database.

Configurations Affected

System has one or more devices connected to the A15 ACP port

Conditions

1) The L2 Data RAM latency is programmed to 4 or more cycles, OR ACP is in use, OR A15 is configured with one or more Data RAM slices
2) Multiple stores are sent from the L1 data memory to the L2 cache
3) Memory type of the stores: strongly ordered, device; normal inner-NC, inner-WT, or inner-WBNoAllocate
4) Multiple evictions from the L1 occur intermixed with the non-cacheable stores
5) The evictions and stores are stalled and hazarded in an extremely rare manner

Implications

Incorrect data will be written to the L2 cache or to memory.

A typical 2-CPU A15 implementation will not use a data slice, and will have a data RAM latency of 3 or less. If ACP is not used, that implementation will not be affected.

If an A15 implementation is using ACP, is configured with a data RAM slice, or is using a slower data RAM, in very rare circumstances data corruption could occur. This issue has been reproduced a single time in a stressful environment with a data RAM latency of 8 cycles. The required boundary conditions become less likely to align with lower latency data RAMs. The issues has not been reproduced in other configurations, but can't be ruled out except in the configurations described above.

The default reset value of the data RAM latency is 8 cycles, and would then be programmed by boot code to the faster value (typically 3-5 cycles) before turning on the MMU or enabling the cache.

Workaround

If a Data RAM slice is configured in the A15, there is no workaround.

If the system uses ACP, ACP masters must not issue memory requests to A15.

Data RAM latency in the L2 should be programmed to 2 or 3 cycles. 4, 5, 6, 7, and 8 cycle data RAMs should never be used. Because the reset value of the L2CTLR[2:0] is 3'b000 (corresponding to 8 cycle data RAM latency), after reset and before the MMU is enabled or the SCTLR.c bit is set, the L2CTLR register data ram latency bits (L2CTLR[2:0]) should be programmed to 3'b001 or 3'b010, corresponding to data RAM latencies of 2 or 3 cycles respectively.

775619: L2 may deadlock or corrupt data if all ways in a single set become blocked from replacement

Category A Rare

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1

Description

A15 has bits in each cache line in the Level 2 Cache (L2) that can block the line from being replaced. These bits are set to indicate a line is currently held in an L1 cache and must be retained, or may be temporarily set if there is a multi-pass request in flight to that line.

It is possible for all 16 lines in a given L2 set to be 'locked'. If this occurs, requests that need to replace a line in the L2 should detect a hazard and retry, waiting for a line to be released. However, there are two scenarios in which all 16 ways becoming locked can cause an issue.

If all 16 ways in a given set are locked and an ACP write-allocate request comes, the ACP request will not hazard correctly. It will replace a locked line, putting the L2 cache in an inconsistent state. This could lead to data corruption to the locked line that was replaced.

If all 16 ways in a given set are locked a cache line fill that needs to replace a line in that set will continuously hazard and restart. In a rare set of arbitration conditions, it is possible that that request will block the other requests that will eventually release the locked/inclusive lines. This will lead to a deadlock.

Note: This erratum matches bug #5298 and 5302 in the ARM internal Jira database.

Configurations Affected

- Cat A Rare for 3 or 4 CPU A15 configurations
- Cat B Rare for 1 or 2 CPU A15 configurations

Conditions

1) All ways in a given L2 set have their lock or inclusion bit set
2) An ACP write allocate request is made to that set OR another cache line fill needs to replace a way in that set

Implications

In an A15 with one or two CPUs, there is an easy zero impact workaround that will prevent all the lock/inclusion bits in a set from ever being set at the same time (see workaround below), completely avoiding the data corruption or deadlock.

In an A15 with three or four CPUs, there is no workaround. A system with no ACP write allocate traffic will avoid the data corruption scenario, but could see a deadlock.

Workaround

This erratum can be completely avoided for an A15 containing only one or two CPUs. The "Force in-order requests to same set/way" bit of the Auxiliary Control Register should be set (ACTLR[23] set to 1'b1) in each CPU. This bit will prevent that CPU from having more than two cache line fills outstanding at a given time to the same set. This will completely prevent the L2 from ever seeing all 16 ways in a set with their inclusion or lock bits set and thus avoid the erratum. Because the current L2 implementation will not process more than two requests at a time to a given L1 data cache set, there is no performance impact to setting this bit.

For an A15 containing three or four CPUs, there is no workaround. Preventing ACP write traffic will prevent the silent data corruption, but there is the possibility of a deadlock.

801819: An eviction from L1 data cache might stall indefinitely in the L2 write buffer preventing a snoop from completing

Status

Affects: Cortex-A15 MP Core -NEON

Fault Type: Programmer Category A Rare

Fault Status: Present in: r2p0, r2p1, r2p2, r2p3, r2p4.

Description

A livelock can occur in the L2 cache arbitration that might prevent a snoop from completing. Under certain conditions this can cause the system to deadlock.

When a cacheable store is executed in the L2 it requires an entry to be allocated in the Fill Evict Queue (FEQ). If an entry is not available the store will restart, as will any younger stores from the same CPU, because stores from the same CPU must execute in order.

If two cacheable stores from the same CPU issue into the L2 just as an FEQ entry becomes available, it is possible for the first store (ST1) to detect the FEQ as full, but the second store (ST2) to see an available FEQ entry and allocate that entry. Both stores are then restarted, the ST1 due to the lack of an FEQ entry, ST2 because the ST1 restarted. This means the FEQ entry temporarily allocated by ST2 is marked for deallocation.

The two stores will then reissue to the L2. If the timing of the reissue aligns with the deallocation of the FEQ entry temporarily allocated previously by ST2, it is possible that ST1 will again detect an FEQ full condition and ST2 will see an FEQ entry available and allocate it. If there is no other activity in the L2 to change the timings, this cycle can repeat until a second FEQ entry becomes available and both stores can proceed, or another FEQ entry is allocated for an unrelated request and both stores stop reissuing waiting for an FEQ entry to become available.

Depending on the L2 cache configuration options there might need to be one or more unrelated stores between ST1 and ST2 in order to hit the required timing for the livelock.

If there is an eviction in the write buffer that is behind the stores then that eviction might be stalled as long as the livelock continues. This stalled eviction might stall a snoop hit on the L1 data cache of that CPU. The stalled snoop might block further FEQ entries from deallocating, either through a dependency in the ACE memory system, or because the FEQ is full of snoop hits to the L1 data cache of that CPU. If all of these conditions occur the system can deadlock.

Note: This erratum matches bug #5433 in the ARM internal Jira database.

Configurations Affected

This erratum does not affect the processor if the revision and variant reported by the MIDR is r2p0, r2p1, r2p2, r2p3 or r2p4 and REVIDR[3] is set to 1.

Conditions

1) The L2 Fill Evict Queue (FEQ) is full.

2) Cacheable write ST1 issues followed by cacheable write ST2 from the same CPU.

3) An eviction is waiting in the write buffer behind ST1, ST2, and one or more additional stores.

4) A snoop (from another A15 core or from the ACE AC channel) is waiting for that eviction to complete.

5) That snoop is preventing further FEQ entries from deallocating.

6) Very specific timing conditions.

Implications

If the erratum conditions are met the part will deadlock.

Cortex-A15 will only issue two types of cacheable stores from a CPU into the L2: Write-Back No-Allocate stores, or Write-Back stores that have been gathered into a full line write using the L1 write streaming logic.

For implementations of Cortex-A15 configured with the “L2 arbitration register slice” (arb-slice) option (typically four core systems) the only cacheable stores that can cause the issue are Write-Back No-Allocate stores. Because Write-Back No-Allocate stores are not commonly used and are easy to avoid, there is a straightforward workaround for these implementations. For configurations with the arb-slice, this erratum is Category B.

For implementations of Cortex-A15 configured without the arb-slice option (typically 1 or 2 core systems) Write-Back No-Allocate stores and streaming cache line writes can both lead to the deadlock, although hitting the conditions with streaming cache line writes is much more difficult. There is still a full workaround, but because it involves disabling write-streaming there is a performance hit on some streaming memory workloads. For configurations without the arb-slice, this erratum is Category A Rare.

Workaround

Do both of the following:

1) Do not use the write-back no-allocate memory type.
2) Do not issue write-back cacheable stores at any time when the cache is disabled (SCTLR.C=0) and the MMU is enabled (SCTLR.M=1). Because it is implementation defined whether cacheable stores update the cache when the cache is disabled it is not expected that any portable code will execute cacheable stores when the cache is disabled.

For implementations of Cortex-A15 configured without the "L2 arbitration register slice" option (typically one or two core systems), you must also do the following:

3) Disable write-streaming in each CPU by setting ACTLR[28:25] = 0b1111

2.6. Category B

774569: Watchpoint may not be taken on second half of unaligned ld/st crossing a 64-byte aligned boundary

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

An unaligned memory transaction that crosses a 64-byte aligned memory boundary is split into two requests in the A15 pipeline. Each of these requests is compared to the active watchpoints to see if a watchpoint event needs to occur.

If the address mask field in the DBGWCR is set to 'No mask' or 'Three bits masked' then the second half of such an unaligned transaction will never trigger a watchpoint.

Note: This erratum matches bug #5275 in the ARM internal Jira database.

Conditions

1) DBGWCR[28:24] (Mask) = 5'b00000 or 5'b00011
2) Unaligned memory request crossing a 64-byte boundary
3) DBGWVR has a watchpoint address in the upper 64-byte memory region of the request

Implications

No watchpoint will be taken.

In most cases, this will not be an issue. If a specific variable in memory is being watched, the lowest byte address of the variable should be set in the WVR and the bug will not occur. Memory requests that are moving a region of memory without regards to the underlying variable locations will tend to be aligned. The issue will only arise when the WVR location specified is not the target address of the memory transaction being issued.

Workaround

Set the DBGWCR[28:24] to 5'b00100 (4-bits masked) or higher. The watchpoint will now be taken correctly. However, Software may have to detect and ignore false triggers of the watchpoint because a watchpoint may also fire for nearby bytes in the same 64-byte aligned memory region as the desired bytes.

774769: Data corruption may occur with store streaming in a system

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1

Description

An L2 write request (A) uses a single data bank for two passes to do an ECC read/modify/write. This allows the last quad word of a cache line transaction (B) to get significantly delayed from the first quad word going through the L2 pipe. During this window the L2 buffer associated with that transaction is deallocated and reallocated to a full cache line streaming store (C) from a CPU that will allocate to the L2 cache without an external memory access. The last quad word of the original transaction then collides with the streaming store and incorrect data is written to the cache for that streaming store.

In order to get the streaming store (C) the CPU must have detected a store stream. A series of cacheable stores in a CPU triggers the store streaming mode in that CPU. The CPU then buffers up 64-byte lines of cacheable store data before sending them to the L2 cache. One of those full line writes misses the L2, but will immediately allocate to the L2 without an AXI/ACE request. This is only possible in a non-ACE system, or with streaming stores to non-shared data. Any shared data in an ACE coherent system would first need to obtain ownership of the line with an AR channel memory request.

Note: This erratum matches bug #5293 in the ARM internal Jira database.

Conditions

1) store streaming enabled and allocating data to the L2 cache
2) a series of incrementing stores is executed to an aligned 64-byte region of cacheable memory
3) either:
   - the memory accessed by the stores is non-shared, OR
   - the system is not coherent over ACE (BroadcastInner and BroadcastOuter pins both deasserted)

Implications

If this condition is hit, external memory may be corrupted.

Workaround

The workaround is to configure write streaming on versions of A15 affected by this erratum such that no streaming-write ever allocates into the L2 cache. This can be done by setting the no-allocate threshold to be lower than the L2-allocate threshold. Once streaming starts, all store streams will go immediately to external memory. ACTLR[28:27] (Write streaming "no-allocate" threshold) should be set to 2’b00 (12 cache lines) and ACTLR[26:25] (Write streaming "no-L1-allocate" threshold) should be set to 2’b01 (64 cache lines) or 2’b10 (128 cache lines).

775621: An eviction behind a store to the on-chip GIC may cause a deadlock in a coherent ACE system

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1

Description

In a coherent ACE system, when a WriteUnique or WriteLineUnique request is issued on the A15 AW channel, it is possible that the request will stall behind a snoop that is issued to A15 through the AC snoop channel. A15 must process the snoop without depending on the completion of the WriteUnique/WriteLineUnique or any AW channel requests issued after it.

In the fail scenario, a WriteUnique or WriteLineUnique is issued on AW and does not receive a BRESP. The system drops AWREADY to prevent further AW channel requests being issued. The system issues a snoop request to A15 on the AC channel.

The dropped AWREADY backs up writes in A15. In order to see the fail, one of the CPUs must issue, in order:

- a store to Noncacheable/Strongly Ordered/Device (NC/SO/Dev) external memory
- a store to SO/Dev memory mapped to the internal A15 GIC registers
- an eviction of an L1 write-back cacheable line

The external store can't exit the write-buffer because it's AWREADY is low. The GIC store can't pass the external store. The eviction should be able to bypass both stores and exit the write buffer, but is incorrectly blocked by the GIC store. Because the eviction can't exit the write buffer, the snoop can't complete. The system will then deadlock.

Note: This erratum matches bug #5301 in the ARM internal Jira database.

Configurations Affected

A15 must be configured with an internal GIC

Conditions

1) ACE system
2) A15 configured with the internal GIC
3) WriteUnique or WriteLineUnique issued on the AW channel
4) AWREADY dropped and held low until an AC channel request completes
5) A CPU issues an external memory NC/SO/Dev store, a GIC store, and an eviction
6) There is an AC channel snoop targeting the eviction

Implications

If the above conditions are met, the system will deadlock.

Workaround

To avoid this erratum, you should execute a DMB instruction between any store to a GIC register and the last preceding store. This could be accomplished with a DMB before a series of GIC register stores, and a DMB after the last store in any exception handler that might return to a thread doing stores to GIC registers.

777770: ESR incorrect for AdvSID HCPTR trapped inst in Hyp mode

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

The EC field of the HSR should be written to 0x7 for an HCPTR trapped coprocessor instruction while executing in HYP mode. A15 treats this as an UNDEF exception and writes 0x0 to the HSR.EC field.

Note: This erratum matches bug #5317 in the ARM internal Jira database.

Conditions

1) HCPTR[10] and/or HCPTR[11] set to 1'b1

2) A VFP/Advanced SIMD instruction is executed

Implications

A hypervisor that is trapping VFP or Advanced SIMD instructions executing in the hypervisor will see 0x0 in the EC field of the HSR. This EC value will indicate an Undefined exception. The hypervisor handler will have to load the instruction that triggered the exception and decode it to determine that the instruction was VFP or Advanced SIMD. This will take somewhat longer than determining the issue directly from the HSR register.

Workaround

If the hypervisor is running with HCPTR[10] or HCPTR[11] set and expecting to trap VFP/Advanced SIMD instructions, ensure that your Undefined exception handler loads the instruction to determine that an Advanced SIMD or VFP instruction was being executed and handle it as needed.

784420: Speculative instruction fetches with MMU disabled might not comply with architectural requirements

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

When all applicable stages of translation are disabled, an ARMv7 processor must follow some architectural rules regarding speculative fetches and the addresses to which these can be initiated. These rules avoid potential reads to read-sensitive areas. For more information about these rules see the description of "Behavior of instruction fetches when all associated MMUs are disabled" in the ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition. Cortex-A15 normally operates with both the MMU and branch prediction enabled. If the processor operates in this condition for any significant amount of time, the BTB (branch target buffer) will contain branch predictions. If both stages of translation are then disabled, but branch prediction remains enabled, these stale BTB entries can cause A15 to violate the rules for speculative fetches.

Note: This erratum matches bug #5360 in the ARM internal Jira database.

Conditions

The erratum can occur only if the following sequence of conditions is met:

1) MMU enabled for at least one stage of address translation
2) Branch prediction enabled
3) Branches executed
4) MMU disabled for all applicable stages of address translation

Note: When executing in a Non-secure PL1 or PL0 mode, for condition 1 at least one stage of address translation must be enabled, and for condition 4 both stages of address translation must be disabled. When executing in any other mode, there is only one stage of address translation, that must be enabled for condition 1 and disabled for condition 4.

Implications

If the above conditions occur, it is possible that after the MMU is disabled, speculative instruction fetches will occur to read-sensitive locations.

Workaround

Branch prediction should be disabled when the MMU is disabled after having been enabled. This should be done by clearing the appropriate Z bit in the System Control register at the same time as or just before the final stage of translation is disabled. Branch prediction should remain disabled until the MMU is enabled, or until the BTB has been flushed. On A15, the BPI* branch predictor maintenance commands will not invalidate the BTB. The BTB can be flushed by setting bit 0 of the ACTLR register, doing any instruction cache invalidate instruction (e.g. ICIALLU), and then clearing bit 0 of the ACTLR register.

784477: CTIINTACK register needs clearing each time it is set

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

The CTI contains a CTIINTACK register, which enables a trigger to be acknowledged through software, instead of using a hardware knowledge using the CTITRIGOUTACK input. The correct operation of this register is that writing a one to the bit corresponding to a trigger output will cause that trigger to be cleared, and this will not affect future triggers.

Because of this erratum, when a bit in the CTIINTACK register is set, it remains set until cleared by writing zero to the register. This causes the corresponding trigger outputs to be acknowledged immediately if they occur again, which can lead to them being missed.

The CTIINTACK register is normally used in two cases:

- To clear a debug-originated interrupt, if required by the interrupt controller.
- To clear a debug entry request generated by another processor, when cross-halting is used.

Conditions

The following conditions must occur:

- A CTI trigger output fires.
- The CTI CTIINTACK register is used to acknowledge the trigger output, by writing a one to the bit corresponding to that trigger output.
- The same trigger output fires again before the corresponding bit in the CTIINTACK register is cleared.

Implications

Trigger outputs might be missed:

- In the case of a debug-originated interrupt that uses CTIINTACK to clear the interrupt, events other than the first event might not cause an interrupt to occur.
- In the case of a cross-halting debug request, after the first time a processor halts and restarts, it might halt without halting other processors with it.

Workaround

This is a workaround for tools vendors.

When the CTIINTACK register is written with a nonzero value, it must be immediately written to again with the value zero. This prevents any future events on the corresponding trigger output from being acknowledged.

If this workaround is used, there remains a race condition, whereby a trigger output occurring between the two register writes might be lost. This is in general not significant, because the timing of trigger outputs and the timing of register writes are not highly correlated, and if the trigger output had occurred before the first register write, then it would also have been lost.

785769: Undefined exception is not generated for LDC/STC instructions which access DCC registers in User mode

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

The Debug Communication Channel (DCC) registers are accessible using MCR/MRC instructions. LDC/STC instructions provide alternate access to DCC registers DBGDTRTXint/DBGDTRRXint.

In Non-debug state when DBGDSCR.UDCCdis is set to 1, then access to DCC register using MCR/MRC and LDC/STC instructions from User mode should generate an Undefined Instruction exception. Access using MCR/MRC instructions correctly generates an Undefined Instruction exception correctly. However access using LDC/STC instructions does not generate the Undefined Instruction exception and incorrectly accesses the register.

Note: This erratum matches bug #5372 in the ARM internal Jira database.

Conditions

1) The processor is in Non-debug state and User mode.
2) DBGDSCR.UDCCdis is set to 1.
3) Either the Hypervisor trap to debug registers in not set (HDCR.TDA==0) or the processor is in Secure state.
4) LDC to DBGDTRTXint or STC to DBGDTRRXint is executed.

Implications

If LDC or STC instructions are executed in User mode, then DCC traffic between debug host and debug target from FIQ/IRQ/Supervisor/Monitor/Abort/Hypervisor/Undefined/System modes can be corrupted due to this errata.

Workaround

Tools must use MCR/MRC instructions to access Debug Communication Channel (DCC) registers from User mode, instead of LDC/STC instructions, in Non-debug state.

Alternatively, software can avoid corruption of DCC traffic by Non-secure User mode code by setting the hypervisor trap for debug register accesses (HDCR.TDA), and handling the LDC/STC instruction appropriately in hypervisor code. There is no software workaround to prevent LDC/STC instructions executing in Secure User mode from corrupting DCC traffic handled in other processor modes.

798181: Moving a virtual page that is being accessed by an active process can lead to unexpected behavior

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

A TLB invalidate instruction followed by a DSB instruction to ensure its completion should remove all uses of the old translation from the system. On Cortex-A15 this might not occur in the following cases:

- The exclusive monitor is physical address based. Moving a memory region in physical memory might cause unexpected results, including multiple threads acquiring the same lock.
- Hazarding logic to guarantee read after read ordering to the same address is physical address based. If a memory region is moved in physical memory ordering violations might occur.
- The DSB instruction might complete before all memory transactions to the invalidated translation are globally observed.

The first two conditions might lead to unexpected ordering and Load-Exclusive/Store-Exclusive instructions behaving in an unexpected way. The third condition might lead to data corruption if the software attempts to access or allows access to the physical memory of the invalidated or moved region before the memory transactions have completed.

Note: This erratum matches bugs #5405, #5407, #5428, #5429 in the ARM internal Jira database.

Configurations Affected

This erratum does not affect the processor if the revision and variant reported by the MIDR is r2p0, r2p1, r2p2, r2p3 or r2p4 and REVIDR[4] is set to 1.

Conditions

1) Software modifies the translation tables to invalidate, restrict the permissions of, or change the physical address of a page or block.
2) Software executes a TLB invalidate instruction to invalidate any TLB entries holding the previous translation.
3) Software executes a DSB instruction to ensure completion of the TLB invalidate instruction and global observation of any memory transactions that depended on the previous translation.

Implications

When software changes the translation tables and invalidates a modified TLB entry:

1) exclusive monitors in the system might be using the old translation
   - this might lead to unexpected Load-Exclusive/Store-Exclusive behavior.
2) the read after read hazard logic might be using the old translation
   - this might lead to a load instruction returning older data than a previous load instruction that accessed the same virtual memory location.
3) outstanding loads or stores to the old translation might not be globally observed
   - this might lead to data corruption if the physical memory of the invalidated memory region is accessed before the memory requests that used the old translation are complete.

Workaround

To ensure the completion of a TLB invalidate, software must first follow the translation table invalidation steps specified in the ARM Architecture Reference Manual:

1) Modify the translation tables in memory as required.

2) Execute a DSB to ensure observation of the stores to the translation tables.

3) Execute one or more TLB invalidate instructions targeting the affected memory regions.

4) Execute a DSB to ensure the TLB invalidate instructions from step 3 have been propagated to all processors in the system.

Next, on the processor that is performing the translation table maintenance, it must also perform these two additional steps:

5) Execute a dummy TLBIMVAIS, meaning a TLBIMVAIS to any address with any ASID.

6) Execute a DSB instruction to ensure the dummy TLB invalidate instruction has been propagated to all processors in the system.

Finally, identify any Cortex-A15 processors in the system that are currently using the translation context of the region being invalidated. For global regions this will be all Cortex-A15 processors not powered down or in reset. For non-global regions this will be all Cortex-A15 processors that are currently using the same translation context (typically ASID and VMID) as the modified translation table entries. On each of those processors, ensure that the following occur (in any order):

- execution of a CLREX instruction
- execution of a DMB or DSB instruction

Once the above sequence is complete all uses of the old translation in the system will removed, and any memory transactions to the old translation will be globally observed.

To meet the final requirement, the processor performing the translation table maintenance must execute a CLREX and a DMB or DSB instruction.

The easiest way to meet the final requirement on other processors in the system is to send an inter-processor interrupt to each required processor. Most interrupt handlers will execute a CLREX instruction (as required by the ARM Architecture on a context switch). The interrupt handler must execute a DMB instruction and then signal completion to the processor doing the translation table maintenance.

Note: this workaround must be implemented on any ARM Architecture processor that is executing TLB invalidate instructions that may affect a Cortex-A15 processor. This would include translation table maintenance being performed on a Cortex-A7 processor when a Cortex-A15 processor is present in the system..

798870: A memory read can stall indefinitely in the L2 cache

Category B
Products Affected: Cortex-A15 MP Core -NEON.
Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

If back-to-back speculative cache line fills (fill A and fill B) are issued from the L1 data cache of a CPU to the L2 cache, the second request (fill B) is then cancelled, and the second request would have detected a hazard against a recent write or eviction (write B) to the same cache line as fill B then the L2 logic might deadlock.

Note: This erratum matches bug #5418 in the ARM internal Jira database.

Configurations affected:

This erratum does not affect the processor if the revision and variant reported by the MIDR is r2p0, r2p1, r2p2, r2p3 or r2p4 and REVIDR[2] is set to 1.

To be affected by this erratum, the “L2 arbitration register slice” configuration option must be included

Conditions

1) The L2 memory system is congested, typically due to requests from other cores, or slow memory responses
2) Four data read requests are issued from one CPU (CPUA) to the L2 and added to the Load Request Queue (LRQ)
3) The four read requests are issued from the LRQ to the L2 pipe, but all stall due to the congestion
4) An eviction or write to line B (write B) is issued from CPUA to the L2 and is added to the Write Request Queue (WRQ)
5) Two speculative cache line fills (fill A and fill B) are issued from CPUA in sequential cycles
6) Fill B is cancelled and the actual fill is not issued
7) Fill B is to the same cache line as write B
8) Write B completes
9) A read request (read C) is issued from CPUA

Implications

If the erratum conditions are met, a false hazard is created between the final read request (read C) and write B (which has completed and deallocated). Read C cannot be issued from the LRQ until a deallocation event is seen for write B. However, as write B has already deallocated, this never occurs. Read C therefore deadlocks. Other cores are likely to stall at the next DSB instruction. Interrupts will not break the deadlock.

Workaround

Write the value 1 to Bit[7] of the L2 Auxiliary Control Register (L2ACTLR[7]: Enable hazard detect timeout). When this bit is set to 1, any memory transaction in the L2 that has been stalled for 1024 cycles is reissued to verify that its hazard condition still exists.

In the erratum case, when the final read (read C) reissues, it determines that no hazard exists and completes normally.

There is not expected to be any negative performance impact from this workaround, as the erratum conditions occur very infrequently.

799270: Writing ACTLR.SMP when the L2 cache has been idle for an extended period may not work correctly

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

If the L2 cache logic clock is stopped because of L2 inactivity, setting or clearing the ACTLR.SMP bit might not be effective. The bit is modified in the ACTLR, meaning a read of the register returns the updated value. However the logic that uses that bit retains the previous value.

Note: This erratum matches bug #5417 in the ARM internal Jira database.

Conditions

1) The L2 cache block has been idle for 256 or more cycles with no memory requests from any core, no external snoops, and no ACP requests.

2) A CPU executes an “MCR p15,0,r0,c1,c0,1” instruction (write the ACTLR register) that modifies ACTLR[6].

Implications

The ACTLR.SMP bit controls whether that CPU should receive distributed virtual memory (DVM) requests (instruction cache maintenance, BTB maintenance, or TLB maintenance commands) from other ARM processors in the system. The processor documentation requires that:

- The only time this bit can be set to 1 is at boot time before enabling the MMU.
- The only time this bit can be cleared to zero is during a specific power-down sequence described in the TRM.

If the errata conditions occur when the ACTLR.SMP bit is being set at boot, the instruction cache or TLB could become incoherent, as that CPU would not receive necessary DVM requests.

If the errata conditions occur when the ACTLR.SMP bit is being cleared during a CPU power-down sequence, the L2 logic may send a DVM request to that CPU when it is powered down or in reset. That CPU would not respond to the request and the system would deadlock.

Note: This erratum does not apply to the r3 versions of Cortex-A15 MPCore processor due to clock gating changes implemented for power savings in those revisions.

Workaround

To avoid this erratum, software must ensure that the L2 logic has been used within the previous 256 cycles before modifying the ACTLR.SMP bit. As specified in the TRM, the only times when this bit can be modified are during boot before the MMU has been enabled, or during a specified reset or power down sequence. When modifying this bit, software must ensure that interrupts are disabled (which may require executing in Secure mode), and then do a memory read to a memory location with Non-cacheable, Strongly-ordered, or Device memory attributes. If the MMU is disabled, all memory will appear Strongly-ordered. A dependency must be created between the returning load data and the MCR instruction that sets the ACTLR.SMP bit, as shown in the code below.

Code sequence for setting the ACTLR.SMP bit:

```
; the following code must be executed with all interrupts disabled
; r1 must contain the value of an Non-cacheable, SO, or Dev memory location or register
;    (typically a memory mapped register with no read side effects would be used)
mrc p15,0,r0,c1,c0,1    ; read current value of ACTLR
orr r0,r0,#0x40         ; set SMP bit (ACTLR[6])
ldr r1, [r1]            ; read a device register (location guaranteed not to hit
                         ; the L1 cache)
and r1,r1,#0             ;
orr r0,r0,r1             ; create dummy dependency between dummy load and MCR to write SMP
MCR p15,0,r0,c1,c0,1    ; Write CP15 ACTLR
```

```
ISB
DSB
```

Code sequence for clearing the ACTLR.SMP bit:

```
; the following code must be executed with all interrupts disabled
; r1 must contain the value of an Non-cacheable, SO, or Dev memory location or register
;    (typically a memory mapped register with no read side effects would be used)
mrc p15,0,r0,c1,c0,1    ; read current value of ACTLR
bic r0,r0,#0x40         ; set SMP bit (ACTLR[6])
ldr r1, [r1]            ; read a device register (location guaranteed not to hit
                         ; the L1 cache)
and r1,r1,#0            ;
orr r0,r0,r1            ; create dummy dependency between dummy load and MCR to write SMP
MCR p15,0,r0,c1,c0,1    ; Write CP15 ACTLR
ISB
DSB
```

2.7. Category B (Rare)

763126: Three processor exclusive access livelock

Category B Rare
Products Affected: Cortex-A15 MP Core -NEON.
Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

In a system with three or more coherent masters that all use the ldrex/strex synchronization primitives to access a semaphore in coherent cacheable memory, there is a possibility of a livelock condition where two masters continuously attempt and fail to get the lock while the third master continuously reads the lock.

This erratum is heavily dependent on a unique set of initial conditions, and upon specific interconnect timing once the livelock has started. It is expected to be rare in a real system that the timing conditions will be hit.

An example: two cores C1 and C2 are contending for a lock using ldrex/strex, and core C3 is looping reading the same semaphore location. Once the livelock condition has started, from the perspective of C1, the sequence will look like this:

1) Execute ldrex, hits the cache in unique state.
2) External snoop takes line to shared state (triggered by C3 read).
3) Execute instructions to process the ldrex result and prepare the strex data.
4) Execute strex, hits cache shared, issues readUnique to bring in line unique.
5) External snoop invalidates line, clearing monitor (triggered by C2 strex that will eventually fail the monitor).
6) Line returns in unique state, but strex fails due to cleared monitor.
7) Loop back to step 1.

C1 and C2 constantly issue ReadUniques due to failing store exclusives that invalidate the line in the other core, each core causing the others strex to fail without making forward progress. No forward progress is made until/unless one of the cores stops (possibly due to an interrupt) or interconnect timing happens to allow enough time for one of them to complete.

NOTE: this erratum is describing additional limitations on exclusives loads and stores in a multi-core system including A15. There is no plan to fix this erratum on future A15 cores, as reasonable code following the ARM architecture guidelines should not be affected.

NOTE: This erratum matches bug #4637 in the ARM internal Jira database.

Conditions

1) One master continuously reading the location of the semaphore.
2) Two masters doing a ldrex/strex loop to the semaphore.
3) Semaphore in write-back shared memory.
4) Three master system (3+ core A15, or 3+ total processors in the system over ACE).

Implications

Neither C1 nor C2 will ever succeed in gaining the lock. Software could stop making progress. An interrupt to one of the cores C1/C2/C3 would likely break the livelock.

Workaround

If there are no more than two coherent masters in the system, no workaround is needed, the issue will not be seen.

The latest version of the ACE specification adds additional command types and system logic to allow processors to avoid this issue. This specification update was not available in time for A15 to take advantage of it and A15 does not implement this ACE feature. As an alternative, A15 installed hardware in each processor to detect that the load/store exclusive livelock scenario may be occurring and delay snoops for a period of time to allow the load exclusive/store exclusive loop to complete and make forward progress. With this fix, no existing code that uses ldrex/strex should need to be rewritten if it follows the ARM Architecture Reference Manual guidelines in the “A3.4 Synchronization and Semaphores” section and is not unreasonably long.

To enable this hardware on Cortex-A15 you must set the "Snoop-delayed exclusive handling" bit in the Auxiliary Control Register, ACTLR[31] to 1. The reset value of ACTLR[31] is 0 for all product revisions r0pX, r1pX, r2pX, r3p0, r3p1 and r3p2.

Note: all references to “ldrex” encompass all Load-Exclusive instructions and “strex” encompass all Store-Exclusive instructions.

771173: Unaligned VLDM larger than 68 bytes to shared cacheable memory hit by snoop can stall core until next interrupt

Category B Rare

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0

Description

A VLDM instruction is executed to shared write-back cacheable memory. The VLDM address is not aligned on an 8-byte boundary, is 72 bytes or greater, and is aligned such that it touches three cache lines (it spans three 64-byte aligned regions of memory). Call the cache lines CL1/CL2/CL3 (from low memory to high memory).

The CL2 cache line is available in the L1 early during the execution of the VLDM, but is evicted (likely by a snoop) after some data has returned, but before the VLDM has completed. CL2 and CL3 are then requested from the L2 to complete the VLDM. CL2 data now returns, but no CL3 data returns. At this point, a snoop request is sent to the L1 for cache line CL2.

The L1 cache will not process this snoop until CL3 returns or the next interrupt. If CL3 is blocked in the system and can't complete until the snoop for CL2 completes, that CPU will stall until the next interrupt.

Note: This erratum matches bug #5249 in the ARM internal Jira database.

Conditions

1) VLDM instruction of greater than 68 bytes, not 8 byte aligned, touches three cache lines, CL1/CL2/CL3
2) Multi-core A15, or external ACE masters actively accessing this memory, generating snoops
3) During execution of the VLDM, after using some CL2 data, CL2 is evicted or snoop invalidated
4) CL2 and CL3 are then requested from the L2, CL2 delivered, CL3 held back
5) Snoop issued to A15 L1 cache for CL2
6) CL3 data not returned with dependency on CL2 snoop completion

Implications

Unaligned VLDM instructions of this size crossing three cache lines should be rare. An analysis of the use of VLDM (looking primarily at GCC and Linux) shows that the vast majority of VLDM usage will be 8-byte aligned (as the stack and the register contexts are 8-byte aligned). VLDM/VSTM memcpy could be used, but has been actively discouraged for general usage as it requires powering up/down the Neon/VFP blocks. Compilers should therefore not produce it. The remaining place where very large VLDMs could exist would be in hand written assembly code. However, if a programmer is taking the effort to hand code assembly, they are likely going for performance, and will therefore very likely align their VLDMs.

Hitting this erratum will required a multi-threaded application using the large unaligned VLDM to access memory that is actively being accessed by another processor at the same time (such that the memory gets snooped at least once and likely twice during the execution of the VLDM).

In a non-ACE system (single A15 cluster), it is very likely that even if the snoop collision occurs, the final cache line (CL3) data will return without a dependency on the snoop. It is possible, however, that a dependency would be created and cause a stall until the next interrupt.

In an ACE system, it will be system dependent how likely it is that the returning load data to CL3 is dependent on the completion of the CL2 snoop.

In general, this erratum is likely to be extremely rare. If it does happen, it will occur as a temporary glitch until the next interrupt and will not be repeated. If in some very unusual code it does occur regularly enough to cause issues, the application or driver that is doing the large VLDM can be modified to avoid the issue completely.

Workaround

The workaround is not to use unaligned VLDM accessing more than 68 bytes in a multi-threaded application where another thread will be actively accessing the same memory location at the same time. Break the VLDM into smaller VLDMs or align the address to 8 bytes. The occurrence of this erratum is expected to be very rare.

773319: Unaligned page boundary crossing load hit by TLB invalidate can stall until next interrupt

Category B Rare

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0

Description

An unaligned cacheable writeback load instruction is being executed. It crosses a page table boundary. Both halves of the load hit in the TLB, and the data required from the lower of the pages misses in the cache. A cache line fill is issued to the L2 cache or to memory. The data required from the upper page misses the cache, but does not issue a cache line fill due to congestion.

While that fill is outstanding, a TLB invalidate is received that knocks out the TLB entry associated with the upper page, but does not affect the lower page. After that TLB invalidate is processed, the cache line fill returns. A new page table request must now be issued to the L2TLB or memory for the upper page.

A snoop request is then received by the L1 data cache targeting the just returned cache line fill. This snoop will not be processed until a the TLB request for the upper page is complete.

If the upper page miss requires a memory read and that memory read is unable to complete until the snoop completes, the processor may stall.

If another TLB maintenance operation is received by the A15 CPU just after the above snoop and before the page miss can be issued, the upper page miss may be prevented from issuing until the snoop completes and the processor may stall.

In either case, the A15 CPU will be unable to respond to the snoop or complete the load until the next branch flush, event flush, or interrupt.

Conditions

1) Unaligned load request that crosses a page table boundary
2) The lower page accessed by the load is to writeback shared memory
3) Both pages hit in the L1 DTLB
4) The required data from the lower page misses in the cache and issues a line fill
5) The required data from the upper page misses in the cache but does not issue a fill
6) A TLB invalidate command from a different CPU is received by the L1 DTLB
7) This TLB invalidate affects the upper page, but not the lower page (The TLB entry for the lower page must stay valid for the erratum to occur)
8) Data returns for the cache line fill
9) A snoop request is received for the cache line that just returned
10) Either:
    1) The page table walk for the just invalidated upper page requires a memory access that stalls in the memory system dependent on the above snoop, or
    2) A TLB maintenance operation from another CPU is received just after the above snoop command

Implications

If the above conditions occur, the CPU will be unable to complete either the unaligned load or the snoop request until the next exception (branch flush, event flush, or interrupt) occurs. No data corruption will occur. The next branch flush, fault, or interrupt will resolve the issue and execution will continue normally.

This should be very rare in real systems and should only have a temporary performance impact when it occurs. A targeted TLB invalidate that affects only one of the two pages being accessed by an active process, combined with the extremely rare timing sequence above, will make this extremely unlikely to be seen in a running system.

Workaround

There is no workaround.

2.8. Category C

770320: Single bit ECC error can cause cache maintenance operation to violate memory ordering

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

If a single bit ECC error occurs in the L1 data cache of A15, the L1 will make a request to the L2 logic to trigger an eviction of that line from the L1 data cache in order to correct the error. This background eviction ignores the normal memory ordering rules on cache maintenance operations. However, if there is a cache maintenance operation in flight at the time the ECC error is discovered, it is possible that this cache maintenance operation will pass a store in the machine to the same address as the cache maintenance operation. This will violate the ARM memory ordering requirements.

Note: This erratum matches bug #5223 in the ARM internal Jira database.

Conditions

1) Store to address A has been executed, has not yet been pushed to the L1 cache or L2 cache
2) DCCMVA or DCCIMVA is executed to address A after the store to address A in program order
3) ECC error detection logic is built into the L1 cache and enabled
4) Single bit ECC error is detected in the L1
5) The DCCMVA or DCCIMVA completes and prepares to go to the memory system after the ECC error is detected and before the ECC error implicit cache maintenance operation is issued
6) Software was depending on the cache being empty or clean at the end of the DCCMVA/DCCIMVA and will fail if the store data was not pushed out

Implications

The DCCMVA or DCCIMVA could pass the store to the same address. This could leave dirty data in the cache where software would expect the cache to be clean or invalid.

This erratum is likely to be rare enough to ignore. ECC errors are inherently very rare. Cache maintenance operations occurring very close to earlier stores to the same address are likely to only exist in specialized code. The odds of a store/CMO/ECC error occurring at the same point in time is very low. In general this erratum can be ignored.

When the erratum conditions do occur, the effect will be silent data corruption. A correctable error will be reported in the syndrome registers, but there will be no way to determine that the rare boundary conditions lined up to allow data corruption. However, because the alignment of events is very rare, this is not expected to add significantly to the silent error rate.

Workaround

In general, this issue should be ignored, other than to understand that affected A15 versions do not have perfect reporting or recovery from single ECC errors.

773023: Order may not be maintained between Strongly Ordered memory requests and Device memory requests on ACE/AXI

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

A15 maintains order between Strongly Ordered (SO) memory requests as required by the ARM architecture memory ordering model. This is done inside the A15 by use of internal ordering logic. On the interconnect, A15 enforces ordering by using the same ARID/AWID value for all SO memory requests from a given processor (guaranteeing read/read and write/write ordering) and by waiting for the completion of all SO and Device memory requests on one channel before issuing SO or Device requests from the same core on the other channel (guaranteeing read/write and write/read ordering).

A15 does the same for Device memory.

However, A15 uses different ARID and AWID for Device memory requests from a given CPU and Strongly Ordered memory requests from the same CPU. Due to this fact, it is possible that the an SO read from a given CPU could pass a Device read from the same CPU and arrive at a single peripheral out of order.

Conditions

1) A system has memory mapped peripherals larger than 4KB
2) Some of the pages mapped to that peripheral are mapped Strongly Ordered and some are mapped Device
3) Software depends upon ordering of these Strongly Ordered and Device memory requests

Implications

This is not an issue for any device that fits in one 4KB memory page, as it is only possible to have a single memory type for that page (SO/Dev aliasing is not allowed).

For larger peripherals, it is possible that Strongly Ordered or Device transactions could arrive at the peripheral out of order.

Workaround

A given peripheral device should be mapped to all Strongly Ordered or all Device memory.

774570: Fault Status bit in register DBGDSCR is implemented as sticky

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

Fault status bit[9] in debug register DBGDSCR might be improperly set for synchronous data aborts in debug state in some cases.

Note: This erratum matches bug #5278 in the ARM internal Jira database.

Conditions

1) Core is currently in debug state.
2) If either of the following occur:
    1) Second stage synchronous data abort occurs in Non Secure PL0 or PL1 mode when external debugger issues an instruction through DBGITR debug register, or
    2) Synchronous data abort other than second stage synchronous data abort occurs in Non Secure PL0 or PL1 mode when external debugger issues an instruction through DBGITR debug register with HCR.TGE bit set to 1.
3) FS bit in debug register DBGDSCR bit gets set due to this synchronous data abort as described above.
4) FS bit is not cleared by external debugger by explicitly writing 1'b0 to bit[9] of DBGDSCR.
5) If any of the following occur:
    1) Another synchronous data abort other than second stage synchronous data abort occurs in Non Secure PL0 or PL1 mode when external debugger issues another instruction through DBGITR debug register with HCR.TGE bit set to 0, or
    2) Another synchronous data abort occurs in Non Secure PL2 mode when external debugger issues another instruction through DBGITR debug register, or
    3) Another synchronous data abort occurs in Secure state when external debugger issues another instruction through DBGITR debug register.

Implications

Read of debug register DBGDSCR fault status bit[9] returns incorrect value as 1 when it should be 0.

Workaround

Whenever the external debugger reads DBGDSCR.FS bit as 1 following a synchronous data abort caused by an instruction issued through DBGITR register, the external debugger should explicitly write 1'b0 to fault status bit[9] of DBGDSCR.

774571: Sampling registers DBGCIDSR and DBGVIDSR can return incorrect value

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

Context ID sampling register DBGCIDSR and Virtualization ID sampling register DBGVIDSR can return incorrect values in some cases.

Note: This erratum matches bug #5279 in the ARM internal Jira database.

Conditions

1) Non invasive debug authentication status currently allows sample based profiling.
2) Debug software access lock status is set as indicated by SLK, bit[1] of DBGLSR.
3) PC sampling register DBGPCSR is read by external sampling agent through external debug interface.
4) Context ID or Virtualization ID has changed and ISB or exception entry or exception return has occurred and PC sampling register DBGPCSR is read by spurious software running on the current cpu or another cpu in the system using memory mapped debug interface before DBGCIDSR and DBGVIDSR registers are read by external sampling agent through external debug interface.

Implications

Reads of debug registers DBGCIDSR and/or DBGVIDSR might return a value which is not consistent with PC value sampled by external agent.

Workaround

This issue can occur only when a spurious read to DBGPCSR is made by software running on a cpu in the system. One possible workaround is to set up a translation table for the system register map such that it is not possible for software to access Cortex-A15 debug register space.

775620: Unaligned load crossing page boundary between different memory types may stall until next interrupt

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

An unaligned load instruction crosses a 4k or greater memory boundary where the upper page is normal cacheable memory (write-back read, write, or read/write allocate), and the lower page is normal memory that will not allocate the cache (non-cacheable, write-through, or write-back no-allocate). The upper part of the data misses in the cache and both parts make requests to memory.

The data for the lower half returns but is not used and must be reissued. This can happen for a variety of reasons including a TLB invalidate of the lower page while the request is outstanding. The data for the upper half returns and is in a fill buffer.

A snoop is now generated for the cache line containing the lower part of the load data. This snoop will not complete until the upper half data has returned.

In a non-ACE system with no external snoops coming into A15 on the AC channel, the upper data will return and execution will continue.

In an ACE system, it is possible that the memory read of the upper half of the data will be stalled in a queue with some resource dependence on a snoop. That snoop may be targeting the lower half cache line, or may be stalled behind the snoop to the lower half cache line. If this is the case, the A15 will stall until the next interrupt.

Note: This erratum matches bug #5300 in the ARM internal Jira database.

Conditions

1) ACE system with A15 BROADCASTINNER or BROADCASTOUTER asserted
2) Load instruction that accesses bytes from two different 4k pages
3) The lower page is Normal Non-Cacheable or Normal Write-Through or Normal Write-Back No-Allocate
4) The upper page is Normal Write-back read, write, or read/write allocate and is inner or outer shared
5) The load instruction is one of the following:
   - ldrh or ldr
   - ldm that is not 8-byte aligned
   - vld* to 32bit Si registers that is not 8-byte aligned
   - vld* to 64bit Di registers that is not 16-byte aligned
6) A snoop is generated to the upper page cache line
7) Upper cache line data has returned, lower line read will not complete due to a dependency on the snoop

Implications

This is not expected to be a significant problem for any real systems. Page crossing between WB and WT/NC is deprecated in ARMv7. Page crossing between WB-allocate and WB-NoAllocate is still supported, but expected to be extremely rare or non-existent in real systems. When a page of memory is requested by an application or driver, the OS will deliver a contiguous section of memory with the same attributes.

Even if such a page crossing load does happen, the odds of another cpu snooping that cache line with the correct timing, and the other half of the load instruction getting blocked behind the snoop are extremely low.

If all of the above conditions are ever hit, the effect would be a stall until the next interrupt. Because of this, as long as the condition occurs rarely (and doesn't occur in a high privilege level with interrupts disabled) the impact will be minimal.

Workaround

The OS should not allocate adjacent virtual memory pages with the upper page mapped as shared normal WBRA, WBWA, or WBRWA memory and the lower page mapped as a normal memory type that will not allocate the cache.

775622: Debug version of Cache Type Register (CTR) IminLine field are incorrectly encoded

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

Debug version of CTR.IminLine, register bits[3:0], are incorrectly encoded as 4'b0100 instead of 4'b0011 when IMINLN input of Cortex A15 is LOW.

Note: This erratum matches bug #5304 in the ARM internal Jira database.

Conditions

This erratum requires both of the following conditions to be met.

1) The IMINLN input of Cortex A15 is LOW. This input is sampled only during reset.

2) The debug register 833 at offset 0xD04, which is an alias of the CP15 Cache Type Register CTR, is read.

Implications

This should not create any issues with an external debugger because the actual value encoded in the register has no effect on hardware operation. CTR.IminLine is used by software to determine stride for cache maintenance operations while operating on a range of addresses. The processor uses the IMINLN input to generate this value. When IMINLN is HIGH,CTR.IminLine is encoded as 4'b0100 indicating 64 bytes. When IMINLN is LOW, CTR.IminLine is encoded as 4'b0011 indicating 32 bytes. This signal does not affect internal instruction cache line size or operation of the cache maintenance commands in hardware. Cortex-A15 hardware always uses 64 bytes as the minimum instruction cache line size.

Workaround

Ignore the CTR.IminLine value in the debug register reads of register 833, the Cache Type Register (CTR). If required, an external debugger can halt the processor and read the Cache Type Register (CTR) using a CP15 instruction to determine the value used by software.

777769: ICache parity error may not be corrected for NC code

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

If an instruction fetch to a non-cacheable page in memory gets a false hit on a line in the instruction cache due to a parity error in the instruction cache tag array, incorrect instructions may be executed.

Note: This erratum matches bug #5312 in the ARM internal Jira database.

Conditions

1) MMU enabled
2) Instruction fetch to page marked SO/Dev/Normal-Non-Cacheable
3) Parity error in instruction cache tag
4) Corrupted tag matches the physical address of the cache line being fetched

Implications

In an A15 configured with L1 parity/ECC and with parity/ECC checking enabled, in very rare circumstances a parity error can cause delivery of bad instructions while executing non-cacheable code. This is not expected to be an issue in normal systems as no normal programs will have instructions in non-cacheable memory with the MMU enabled. At boot, or any other time that the MMU is disabled, the erratum will not occur.

Workaround

Place instructions in cacheable memory whenever possible. If you must run non-cacheable code with the MMU enabled, first invalidate the instruction cache.

777771: Ordering of read accesses to the same memory location may not be ensured in the case of an unaligned load

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1

Description

An unaligned 64-byte boundary crossing load hits a cache line currently being returned from memory in one of the L1 fill buffers and returns its data to a register, executing out of order ahead of load instructions that are earlier in the program. One or both of the cache lines touched by the load are then lost from the L1 data cache (due to eviction or snoop invalidate). Another master modifies the lost cache line or lines. One of the earlier load instructions (which need not be unaligned) executes, reads the modified data, and returns it to a register. A15 should detect that the earlier load has returned more recent data (in violation of the ARM memory ordering model) and flush the younger load. In this case it does not correctly detect the hazard.

If this erratum is hit these loads are observed out of program order by the master doing the stores. This violates the ARM memory ordering model.

Note: This erratum matches bug #5328 in the ARM internal Jira database.

Conditions

1) Unaligned load executes out of order ahead of loads that are earlier in program order
2) The unaligned load touches two 64-byte aligned regions and is one of the following:
   - ldrh or ldr
   - ldm that is not 8-byte aligned
   - vld* to 32bit Si registers that is not 8-byte aligned
   - vld* to 64bit Di registers that is not 16-byte aligned
3) At least one of the lines read by the unaligned load is evicted from the L1 data cache (snooped or replaced)
4) The evicted line is modified by another master
5) One of the earlier loads now executes, brings in the modified line, and returns one of the modified bytes

Implications

In general, when one master is modifying a memory location and another is reading from it, the accesses will be synchronized with the appropriate use of semaphores or locks such that the write to memory is complete before the read is executed. Such code will not be affected by this erratum.

However, certain lock-free synchronization techniques depend on the memory order property that earlier loads will not return more recent data for a given byte in memory than later loads. If an unaligned 64-byte boundary crossing variable is used in lock-free programming it may not work correctly.

This is expected only to happen in hand-written assembly code. In a compiler, variables that are used for lock-free synchronization must be marked as ‘volatile’ to avoid normal compiler reordering. Current compilers will not place volatile variables in an unaligned memory location and will not generate code that will be affected by this erratum.

Workaround

Whenever possible, use aligned memory locations for any memory reads that will rely on in order observation by other masters. Volatile variables in the ARM and GNU compilers will meet this requirement for volatile variables.

If there is an unaligned memory read (as defined in condition 2 above) that must be observed in order with earlier reads, place a DMB instruction before it. This will correctly enforce the observation order.

777772: Device LDM may stall if memory type changed asynchronously from Device to Normal

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

A LDM/VLDM or extension register element load instruction begins execution. This load will be broken up into multiple micro-ops of 8 or 16 bytes. The first few load micro-ops hit the TLB and return translations with the memory type Strongly Ordered or Device. Before the rest of the micro-ops in the load instruction can complete, a TLB invalidate instruction from another master removes that TLB entry. The LDM therefore triggers a new tablewalk. This tablewalk returns a translation with a memory type of Normal. The CPU will be unable to complete the load. The CPU will stall until the next interrupt. If this occurs in the highest level of privilege with interrupts disabled, the CPU may deadlock.

Note: This erratum matches bug #5329 in the ARM internal Jira database.

Conditions

1) A load instruction of greater than 8-bytes is executed
2) The current translation for the targeted page in memory is Strongly Ordered or Device
3) A TLB invalidate from another CPU is received in the middle of executing the large load instruction
4) The tablewalk for the invalidated page returns a Normal memory page (NC, write-through, or write-back)

Implications

If all the conditions occur, the CPU will stall until the next interrupt.

This is not expected to be an issue in any real systems. An OS will not remap a page from an SO/Device type to normal memory while an active process is accessing that memory.

If in a rare case the OS does shift an actively accessed page from Strongly Ordered/Device to a Normal (NC/WT/WB) memory type, it is possible that the thread accessing the page may stall until the next interrupt. As this is expected to occur extremely infrequently or never, the impact will be low.

Workaround

When remapping a virtual address page from Strongly Ordered or Device to a Normal memory type, the OS should first remap the page from the original memory type to an invalid page, execute a DSB to ensure it is complete, and then remap the page as Normal memory. This will avoid this issue.

777774: DCCMVAC/DCCSW colliding with ReadOnce from ACE could cause data corruption

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

A cache maintenance instruction (DCCSW to the level 2 cache or DCCMVAC) is executed to a line that is dirty in the A15 L1 data cache or L2 unified cache. In a coherent ACE system, this will cause A15 to generate an ACE WriteClean operation pushing out the dirty data while retaining the line in the cache in Unique Clean state. The cache state is updated and the dirty data is placed in a buffer to be sent over the ACE write channel.

A ReadOnce command for the same line is then received over the AC snoop interface. The L2 cache detects that the WriteClean is in the buffer and sends the dirty data out over the CDATA channel in response to the snoop, marking the data as dirty. A15 has no completed the snoop and the line is in UniqueClean state. The interconnect does not yet finish writing the dirty data back to memory, however.

A write from an A15 CPU or ACP then modifies that line in the L2 cache, changing its state to UniqueDirty. The line is then replaced in the L2 cache and a WriteBack is issued on the ACE write channel.

It is possible that this WriteBack will arrive at memory before the memory update generated from the CDATA snoop response. If this occurs, the most recent WriteBack data will be overwritten by the older CDATA data. This will lead to stale data in memory.

To avoid the above case, the ACE protocol states that a UniqueDirty line can’t transition to UniqueClean in response to a ReadOnce. A15 violates this restriction.

Note: This erratum matches bug #5334 in the ARM internal Jira database.

Conditions

1) Cache line in Unique Dirty state (Modified) in A15 L1 data or L2 cache
2) DCCSW or DCCMVAC instruction executed to that line
3) Before the WriteClean command is issued to the bus, a ReadOnce occurs for that line
4) The dirty data is sent on the CDATA channel in response to the snoop, but stalls in the system
5) That cache line is further modified by A15
6) That cache line is cleaned or replaced in the A15 L2 and generates a WriteBack or WriteClean
7) The WriteBack or WriteClean completes to memory before the stalled CDATA channel response above completes

Implications

If the above conditions are met, the latest data from A15 to that line will be overwritten by the stale data in the CDATA response. This is expected to be very uncommon or impossible in real systems, however.

A ReadOnce command will be received by A15 if a non-caching master is making a read request for that line. This would tend to imply that cache coherence for that memory location is being handled by the ACE hardware coherence mechanism.

If a DCCMVAC instruction (Data Cache Clean by MVA to point of coherence) is executed for a given memory location, there is software cache maintenance occurring. Typically software would be pushing data out of the L1/L2 caches so it can be seen by an external master that is not participating in ACE hardware coherence.

A DCCSW instruction is generally used only as part of loop cleaning all dirty data out of the caches in preparation for powering down A15.

This erratum can only occur if a ReadOnce occurs concurrent with one of the above software clean instructions. In the case of the DCCSW power down sequence, it is possible that a non-caching master would generate a ReadOnce while a DCCSW is executing. However, even if this occurs, during the core powerdown operation software will typically have the caches disabled and will be making no further modifications to those lines. There will be no second modification of the line to generate the second WriteBack/WriteClean required by the race condition.

In the case of software communicating with an external peripheral through a cached memory buffer, this erratum is expected to be rare for two reasons. First, generally only software or hardware coherence will be used for a given memory location, not both. This means that getting a ReadOnce and a DCCMVAC to the same location concurrently

would not be expected. Second, software will generally modify a section of memory and then clean that memory from the caches to be used by the non-hardware-coherent peripheral. While that peripheral is using the data, it will be unusual for software on the A15 to further modify that memory location. Because of this, the second WriteBack/WriteClean required to generate the race condition will be unlikely.

DCCMVAU (clean to point of unification), often used for self-modifying code, will not trigger this erratum.

Workaround

Do not execute DCCSW or DCCMVAC instructions to cache lines that might be accessed by an ACE ReadOnce command while A15 is actively modifying it. Replace any DCCSW that are not part of a powerdown sequence with DCCISW (clean/invalidate). If cache clean by MVA (DCCMVAC) must be done to a region of memory being accessed by a coherent non-caching peripheral, do a clean/invalidate by MVA (DCCIMVAC) instead.

780121: PTM might not acknowledge a trace flush request when cpu is in WFI.

Category C
Products Affected: Cortex-A15 MP Core -NEON.
Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

When a trace sink requests a flush of a trace source by asserting AFVALID, the trace source normally acknowledges the flush request by asserting AFREADY after all the trace in FIFO has been output.

Under certain conditions, the PTM might not acknowledge a trace flush request even after all the trace has been output.

Note: This erratum matches bug #5346 in the ARM internal Jira database.

Conditions

The following sequence must occur:

1) A processor power on reset or debug trace reset request
2) The PTM is enabled
3) The PTM is later disabled
4) A WFI or WFE instruction is executed
5) The processor stops its internal clocks
6) The processor is still powered up

In the above scenario after step 4, PTM will de assert AFREADYM temporarily for few processor clock cycles and few trace clock cycles. If the processor clocks are stopped before AFREADYM is asserted again, then any new flushes will not be acknowledged until the processor clocks are restarted.

AFREADYM will be correctly asserted during WFI in any of the following cases:

- The processor is powered down and clamps are activated
- The PTM is reset
- The PTM was never enabled out of trace reset
- The PTM is not disabled
- The PTM is disabled during WFI

Implications

If a trace sink initiates a flush request and then stops on the flush completion, then due to this erratum the flush never completes and the trace sink will not stop.

If the PTM is connected to a CoreSight trace funnel, then if a flush request is initiated the funnel will wait for the all trace sources to acknowledge the flush before continuing. This erratum might cause the funnel to stop accepting new trace from all other trace sources, preventing any further trace capture.

Workaround

There are two workarounds:

- If the PTM is connected to a trace funnel, after disabling the PTM, the trace analyzer must disable the corresponding trace slave port in trace funnel. This will make the trace funnel acknowledge the flush immediately.
- Instead of disabling the PTM, the PTM should be configured to generate no more trace, but must remain enabled.

784419: An unaligned load crossing a 4K boundary between SO/Dev memory and WB memory can stall the core until the next interrupt

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

If an unaligned load crosses a 4k page boundary between two pages where the lower page is mapped to Strongly-ordered or Device memory and the upper page is mapped to Normal, Write-Back Cacheable memory, in certain rare cases the core may stall until the next interrupt. A memory transaction that crosses a boundary between these page types is architecturally UNPREDICTABLE and should not occur in any real code. However, if this were to occur at the highest privilege level with interrupts disabled, it could deadlock that CPU.

Note: This erratum matches bug #5355 in the ARM internal Jira database.

Conditions

1) A load instruction is executed that accesses bytes from two different 4k pages
2) The lower page is Device or Strongly-ordered memory type
3) The upper page is Normal Write-Back Cacheable memory, that is also Read-Allocate, Write-Allocate, or both
4) The load instruction is one of the following:
   - LDRH or LDR
   - an LDM that is not 8-byte aligned
   - a VLD* to 32bit Sd registers that is not 8-byte aligned
   - a VLD* to 64bit Dd registers that is not 16-byte aligned

Implications

If the above conditions occur, it is possible that in rare cases the core will stall until the next interrupt. If this occurs in a situation where no interrupt will occur, the CPU deadlocks. Examples of where a deadlock might occur are if there is no timer interrupt in the system, or the conditions occur at the highest privilege level with interrupts disabled.

Workaround

Do not execute loads that cross between Strongly-ordered or Device memory and Normal memory pages. Architecturally, such a load is UNPREDICTABLE.

784469: CTI Authentication Status register is incorrect

Category C
Products Affected: Cortex-A15 MP Core -NEON.
Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

The AUTHSTATUS register is a read-only register in the CTI that reports the debug level supported by the CTI and the current status of the debug level.

The CoreSight Architecture Specification specifies bits [3:0] in the AUTHSTATUS register as below:

- [3:2] Non-Secure Non-Invasive Debug
- [1:0] Non-Secure Invasive Debug

For each of these fields, the value of the status bits as returned by the CTI and their meanings are specified as below:

Value Description

2'b10 Functionality disabled

2'b11 Functionality enabled

In the CTI each pair of bits ([3:2] and [1:0]) in the AUTHSTATUS register currently read:

- When functionality is disabled - 2'b01

but should read (as per table above):

- When functionality is disabled - 2'b10

The bits are swapped.

Condition 1

AUTHSTATUS[1:0] - Non-secure Invasive Debug

- DBGEN input to the CTI is LOW
- AUTHSTATUS register is read

Condition 2

AUTHSTATUS[3:2] - Non-secure non-Invasive Debug

- NIDEN input to the CTI is LOW
- DBGEN input to the CTI is LOW
- AUTHSTATUS register is read

Implications

The status of the debug level supported by the CTI as returned by the AUTHSTATUS register read is incorrect. The masking of trigger inputs and outputs using DBGEN and NIDEN is not affected by this erratum. The return of an incorrect value might lead to incorrect operation of debug tools.

Workaround

This is a workaround for users and tools vendors. When reading the AUTHSTATUS register, swap the bits in the affected fields and interpret the read data accordingly.

788419: If a single memory address is aliased as both shareable and non-shareable, a CMO to this same line might deadlock

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r2p0, r2p1, r2p2, r2p3, r2p4

Description

If an invalidating data cache maintenance by MVA operation (DCIMVAC or DCCIMVAC) issued by an A15 CPU hits a dirty line in the L2 cache and collides in a narrow window with a ACE snoop to the same line, the data provided as part of the snoop response might be corrupted or the system could deadlock.

The cache maintenance operation allocates a data buffer to hold the dirty data to be evicted. The snoop misses in the cache, but is detected as a hit on the eviction that is in progress. The eviction data is converted to a snoop response, and the snoop becomes associated with the dirty data in the data buffer, instead of being associated with the cache maintenance operation data buffer entry. For one cycle during this conversion when the cache maintenance operation is assigning the data buffer to the snoop, that data buffer appears to be available for an unrelated new request. If a new operation that is in the pipeline during that one cycle is allocated to the data buffer, and this allocation overwrites the data before the data is read out for the snoop response, then the snoop data is corrupted.

Note: This erratum matches bug #5378 in the ARM internal Jira database.

Conditions

This erratum requires the following sequence of conditions:

1) The same physical memory address must be aliased as both shareable and nonshareable in two different page table entries.
2) A DCIMVAC or DCCIMVAC instruction is executed on an A15 CPU to cache line A.
3) Cache line A is in the L2 cache and is dirty.
4) An ACE snoop to line A occurs.
5) The ACE snoop misses the cache, because the invalidation has already occurred, but matches on the eviction in flight before the eviction is committed to the ACE write channel.
6) The ACE snoop data response is held up, but the cache maintenance operation for cache line A is issued on the AR channel, completes, and gets an R channel response.
7) The cache maintenance operation deallocates from the L2 cache buffers before the snoop data is sent out.

Implications

The erratum conditions require the CMO to be issued after the snoop has been received, but to complete before the snoop completes. This can occur when there is disagreement on the shareability of physical address A. A15 issues a CMO to address A as non-shared memory, but another master has issued a shared memory request to A which trigger the snoop. These transactions may go through different paths in the memory system and not hazard. Disagreeing on the shareability of memory is unpredictable in the ARM architecture and therefore the erratum cannot occur with correctly programmed page tables.

Workaround

For all systems:

- Ensure that all masters in the system agree on the shareability of all memory locations and that the interconnect will not issue snoops to A15 for non-shareable memory locations.